Interactive Graphics - Final Project

Andreea Nica – 1593889

25/07/2021

Hungry Duck

This project consists in the implementation of a videogame which is a “non-stop running” kind and its objective is to avoid obstacles and reach the highest score possible. The goal of the user is move the duck in order to take the food and avoid the obstacles. Everytime Duck catches a food, it gains 5 points. Every 100 points, level is incremented, such as game speed and difculty. Duck has 3 lives and everytime it catches a danger, lives are decremented by one. After three lost lives, game is over. Duck's movement are controlled through up and down mouse movement . The user can pause and resume the game in every moment pressing the space bar. There is a music for the game that can be muted or regolated in the volume.

Environment

Three.js is a Javascript library for 3D web computer graphics based on WebGL. The latter in turn, is a low level 3D graphics library that allows OpenGL to be executed on the browser. Three.js in fact, is a high level implementation of OpenGL, hiding the pragmatic and time consuming coding of gl programs, vertex-shader, fragment shader, buffers, projections and rendering. Including the library to the project is just importing into the index.html the file .The four three.js fundamental objects that manage and set up the webGL environment are:

1. THREE.Scene

2. THREE.Object3D

3. THREE.Camera

4. THREE.WebGLRenderer

A more specific child of the object3D is the THREE.Mesh that links a geometry with the material properties, giving the final object with all the needed buffers in order to be rendered with vertices, textures and colors.

The fundamental properties of an object3D, used the most for manipulating it are:

- modelViewMatrix: it is passed to the shader for computing the position of the object
- normalMatrix: it is passed to the shader and used to calculate lighting for the object
- position: a Vector3 representing the object's local position
- rotation: object's local rotation (Euler angles), in radians.
- scale: object's local scale

The camera is responsible for handling the projection matrix, used to define the view volume of the scene. For this project the perspective projection was chosen, because it is designed to mimic the way the human eye sees, manipulating the fovy, aspect, near and far parameters. Finally, the WebGL renderer, the most important component of the Three.js library, is the one responsible for linking the shaders with the gl programs, setting up the 3D graphics in the canvas element, and it is in fact called to render with the requestAnimationFrame function as follows: renderer.render(scene, camera).

Geometry

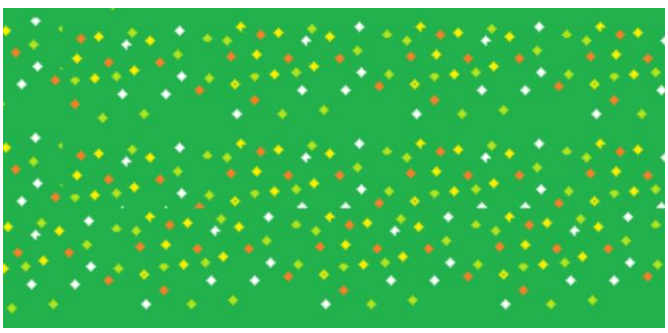
The Three.js library allows us to build geometries easily, based on the one we need. The geometries I built in the project are:

Duck

The first hierarchical model present in this project is Duck composed by body, neck, head, legs, tail, eye, beak and wings. I applied to all the components a PhongMaterial shader to give the wanted effect of material (color, opacity, transparency and so on). Moreover, all the components are made up with a mesh function of geometry and material. Duck model is set to emphasize the shadow given by the directional light. The body is an oval object built with SphereGeometry of Three.js and then scaled. It is the principal component of the hierarchical model. Then I positioned the neck as BoxGeometry. The head is a spherical object (always SphereGeometry) that has been positioned ahead of the neck. On the head we can find the eye (SphereGeometry), and the beak(CylinderGeometry). Going ahead, we find the back and front legs, realized with BoxGeometry and the tail (CylinderGeometry).

Grass

The game environment includes a cylinder that rotates around its z-axis and will be the ground and let the position and orientation of the player be the same, giving the illusion of a real flying animation. To the cylinder is applied a texture, configured in the specific function congureTextureGrass(), in order to give to the cylinder the aspect of real field through Three.js TextureLoader function.



Three

Another hierarchical model consisting in a cylinder mesh as trunk (root) and four cylinder meshes as crowns, each one child of the previous one. Trees are distributed over ground dened in the Sky object.

Fish

Fish creation is made through createFish() function, that simply call the function FishOwner(num). FishOwner(num) function uses two arrays (fishInScene[] and fishStock[]). First of all, the function add to fishStock[] the prexed number of figures. Then, it add figures to the scene with fishInScene[] array, respecting distance and frequence parameters. These operations are made in FishOwner.prototype.addFish() function.

Obstacle

Obstacles are created with the same mechanism as Fish, i.e. calling the function createObstacle(), that in turn creates an instance of ObstacleOwner. ObstacleOwner has two implemented functions, i.e. addObstacle() and ObstacleAnimation(), that manage the creation and the animation of dangers.

To the sphere is applied a texture, configured in the specific function congureTextureObstacle(), in order to give to the spheres the aspect of real field through Three.js TextureLoader function.

Lights

In this project, I implemented three lights: an ambient light, an emisphere light and a directional light.

The **ambient light** globally illuminates all objects in the scene equally.

The **hemisphere light** is used to shade from the sky color to the ground color, creating an effect of gradient color, typical of the skyline.

A **directional light** is a light that gets emitted in a specific direction. This light will behave as though it is infinitely far away and the rays produced from it are all parallel. The common use case for this is to simulate daylight and that was the use in the project presented. This light can cast shadows and the model used for the shadows was the Phong model.

To make our environment more realistic we have added a Day/Night cycle. In a first approach I created a function that varied the intensity of hemisphere light, but the objects were too dark so, I decided to vary the background colour in the css file giving the illusion of day/night. In this way the objects (duck, fish and obstacle) rested illuminated.

```
#scenario {
        position: absolute;
        width: 98%;
        height: 98%;
        overflow: hidden;
        background: #add1f3;
        animation: mymove 180s infinite;
}

        @keyframes mymove {
        50% {background-color: rgb(4, 4, 63);}
  }
```

However, to create a light animation, I vary the position of the directional light as a function time.

I used Math.sin/cos on x and z-axis.

```
var time = Date.now() * 0.0005;
directionalLight.position.x = (Math.cos(time*0.05) *100);
directionalLight.position.z = (Math.sin(time*0.05) *100);
//hemisphereLight.intensity = (Math.cos(time*0.05) *100);
//hemisphereLight.intensity = Math.min(0.1);
```

Animation

In the function updateDuck() there are updates of wings and tail motion and function move() is called. In move() function, we deal with the motion of the duck throughout the y-axis. . A mouse listener (placed in init() function) gives the coordinates for dislocation on the y-axis. The duck has rotations on x and z axis too. Moreover legs rotate along with the body, using the dierence between the mouse position and the duck position and exploiting the hierarchical model.

The fish animation is managed in the function FishOwner.prototype.foodAnimation(). In the same function, I manage the moment in which the duck catches the fish, with the corresponding increment in points.

In function obstacleAnimation(), there is also the managament of the situation in which duck bumps against a danger. In this case, a life is lost and the table in the scene is updated through the function updateLife().

Final Results